2008

An evaluation between Bloom Filter join and PERF join in Distributed Query Processing

Ming Pei
University of Windsor

AN EVALUATION BETWEEN BLOOM FILTER JOIN AND PERF JOIN IN DISTRIBUTED QUERY PROCESSING

By
Ming Pei

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2007

Your file Votre référence
ISBN: 978-0-494-42276-2
Our file Notre référence
ISBN: 978-0-494-42276-2

Abstract

Nowadays, with the explosion of information and the telecommunication era's coming, more and more huge applications encourage decentralization of data while accessing data from different sites [HFB00]. The process of retrieving data from different sites called Distributed Query Processing. The objective of distributed query optimization is to find the most cost-effective of executing query across the network [OV99].

Semijoin [BC81] [BG+81] is known as an effective operator to eliminate the tuples of a relation which are not contributive to a query. 2-way semijoin [KR87] is an extended version of semijoin which not only performs forward reduction like traditional semijoin does, but also provides backward reduction always in cost-effective way. Bloom Filter[B70] and PERF [LR95] are 2 filter based techniques which use a bit vector to represent of the original join attributes projection during the data transmission. Compare with generating a bit array with hash function in bloom filter, Perf join is based on the tuples scan order to avoid losing information caused by hash collision.

In the thesis, we will apply both bloom filter and perf on 2-way semijoin algorithms to reduce transmission cost of distributed queries. Performance of propose algorithms will compare against each others and IFS (Initial Feasible Solution) through amount of experiments.

Keywords:
Distributed Query Processing, Semijoin, Bloom Filter, Perf Join.

Dedication

To my family

Acknowledgements

First I would like to thank my supervisor, Dr. Joan Morrissey, for her academic advice, for her support and help throughout my master study. She has guided me in my whole master study, got me interested in the research, and carefully reviewed each update of the thesis. I would like to express my admiration for her dedication to teaching and research.

I would also like to thank my external reader, Dr. Yuntong Wang, my internal Reader, Dr. Jianguo Lu, and my thesis committee chair, Dr. Robert Kent for their great suggestions and valuable time given to review my thesis.

I would like to thank Dr. Liwu Li, for his guidance. His sudden and unexpected death was a shock to everyone. He is sincerely missed.

I also should give my thanks to my managers and colleagues in my work who gave me lots of encouragement during my thesis composition period.

Table of Content

List of Figures

Chapter 1 Introduction

Database system once was built centralized to meet the needs of structured information. The increasing demand for efficient means of accessing data coupled with the need to manage increasingly large volumes of data has made distributed relational databases critically important in modern IT systems. Distributed relational database was brought into reality to achieve the advantages of performance, reliability, availability, and modularity.

A distributed database system is defined as a network which consists of processors (nodes) located dispersedly but interconnected to each other via communication channels [V02]. Distributed database is stored on several computers and each site varies in size and complexity. The sites connect to each other via network but self-maintained locally. An essential feature of distributed database is to allow users to access the data at the same time from geographically disperse locations and to retrieve target data set by means of queries.

Distributed Query Processing is the procedure that retrieves data from different sites [AHY83] [HF01]. To run a query in a distributed database, each site processes the query and returns the results to the final query site as an answer. Thus, query optimization becomes a major issue of distributed query processing. The objective is to find the most cost-effective way to execute query over the network. Typically, a given distributed query is processed in three phases as shown in [RK91] [KR87] [TC92] [BRP92] [BRJ87] [CL84]:

(1) ***Local processing phase:*** Selections and projections are performed at local nodes on the joining and target attribute.
(2) ***Reduction phase:*** A sequence of semijoins is used to reduce the size of a relation cost-effectively, with a result of a decrease cost of data transmission.
(3) ***Final query processing phase:*** at query site, the final query will be performed after all relations involved in the query are transmitted to this site.

Due to the local processing costs are negligible by comparing with the communication costs of data transmission. The principal challenge is to design and develop efficient query processing strategies to minimize the communication cost focus on the phase (2) and (3). Semijoin, who acts as powerful size reducer in phase 2, only transfers parts of relations during the distributed query processing against sending the whole relation as join does. Lots of heuristic algorithms are proposed based on semijoin.

As an extended version of semijoin, 2-way semijoin [KR87] not only performs forward reduction as the traditional semijoin operator does, but also provides backward reduction in an always cost-effective way. By using a bit array to represent the join attribute projection, filter technology can achieve the same result as a semijoin but at much lower cost. Two filter-based techniques which are bloom-filter join (Hash Semijoin) [TC92] and perf (Partially Encoded Record Filter) join [LR95] will be discussed in detail afterward. Bloom-filter uses a search filter which is generated with hash function to represent the semijoin projection, while perf join minimize the cost of backward reduction in 2-way semijoin by sending a scan ordered bit array. They have similar storage and transmission efficiency. However, with bloom-filter, semijoin may encounter losing join information as a result of hash collusion. Regardless of existing weakness, these 2 kinds of techniques are most powerful reducers with significantly cheaper transmission cost.

In this thesis, two filter based approaches will be implemented by applying Bloom Filter and PERF into a 2-way semijoin based algorithm. We will evaluate the reduction effect between two algorithms and IFS strategy by amount of experiments. The rest of thesis is organized as follows: Chapter 2 reviews the back ground of query optimization, several operators and core algorithms are presented; Chapter 3 does description of bloom-filter and perf join algorithm respectively; Chapter 4 evaluates the performance of two algorithms by experiments; In Chapter 5, we make final conclusion and future work.

Chapter 2 Background Review

2.1 Definition and Notation

In query processing, we need use the following notation and definition to construct and describe a query strategy:

Projection: The projection of relation R on a set of attributes A is denoted by R[A]. It is obtained by discarding all columns of relation R that are not in A and eliminating duplicated rows if necessary.

Selection: The selection of these tuples whose A-attribute values equal to a specified constant in relation R is denoted by R.A=constant (operators other than "=" e.g., $\geq$ and $\neq$ are also allowed)

Benefit: the data reduced by semijoin,

Net benefit: the value of benefit minus cost, if the net benefit is greater than 0, we call it is a cost-effective.

Schedule: the cost of data transmission used for reducing an involved relation and the transmission of the reduced relation to the query computer

2.2 Cost Model

In distributed query processing, query optimizer considers all the possible ways to execute a query and decides on the most efficient way based on the cost. Cost-based query processing assigns an estimated "cost" to each possible query plan, and chooses the plan with the smallest cost. Costs are used to estimate the runtime cost of evaluating the query. Total time and response time are two cost models used to measure the query execution plan. The total time cost model is the sum of the every single operation happened during the query processing; while the response time model is the elapsed time from the query initiation to the it's end. [AHY83]

The execution cost of a query involves both I/O and CPU spending for local processing at each participating site and communication cost between the networks. It is so happened that network transmission cost is relatively more significant comparing to local processing time. To simplify the problem, typically, local processing is considered negligible and transmission cost is stressed as the major concern. Data transmission cost from one site to another can be represented as a linear function:

$CT = T_{mrg} + T_{tr}$

T_{mrg} is the fixed time for initialization, while T_{tr} is the time of data transmitted from one site to another.

2.3 Join Operation

Join, one of the essential operations, retrieves data from different site and relations. It is a common yet highly time-consuming query operation. As shown in Figure 1, given relation R_1 and relation R_2 on attribute A, a join of R_1 and R_2 is denoted as $R_1.A= R_2.A$ [YC84], where R_1 and R_2 are joining relations and A is the joining attribute. Join is obtained by concatenating each row of R_1 with each row of R_2 wherever the A-attribute values of the 2 rows are equal. Typically, in distributed system, it is very likely that the two relationships are not in the same site. In that case, sequence of operations will be applied to optimize queries. Usually the comparatively smaller sized relation will be transferred to remote site for join operation.

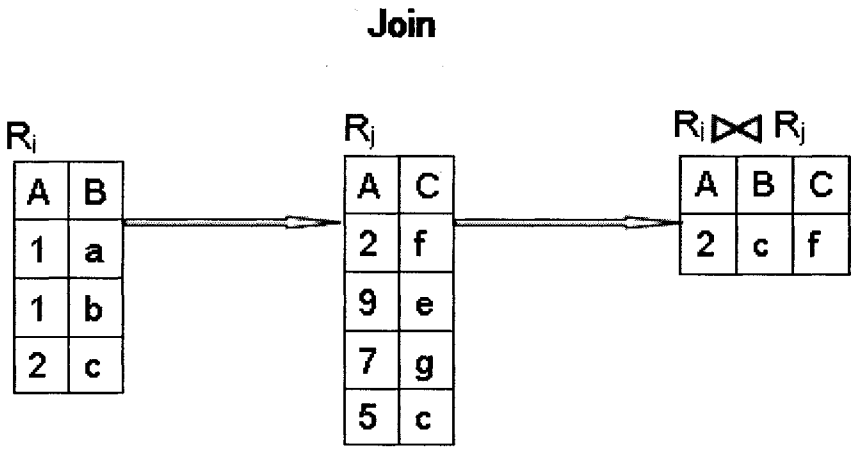

Figure 1: Example of Join

However, some problems may rise during this processing. One is that result relation can be greater than total participating relations and therefore actually increase the cost instead of reducing it. Another problem is that network resources are largely wasted by transmission of an entire relation, while only a small percentage of tuples are required.

2.4 Semijoin Operation

As discussed in the previous chapter, some problems exist in join operator. Therefore, the theory of semijoin is introduced in late seventies. One of the objects of semijoin is to reduce inter-site communication cost. As a relation algebraic operation, semijoin selects a set of tuples in one relation that match one or more tuples of another relation on the joining domains in [BC81-1].

A semijoin from relation R_1 to relation R_2 on attribute A is denoted by $R_1 - A \rightarrow R_2$, where R_1 is the sending relation, R_2 is the reduced relation, and A is the joining attribute. It obtained by shipping $R_1[A]$ which is the projection over attribute A of R_1 to the site where R_2 resides, then make join with $R_1[A]$ and R_2.

Here in the following instance, a semijoin from relation R_i to R_j on attribute A is denoted as $R_i - A \rightarrow R_j$. Relation R_i and R_j resides on different sites respectively.

The result of this semijoin is the projection on the attributes of R of the join of R_i and R_j. There are 2 steps during one semijoin operation:

(1) Send $R_i[A]$ from site R_i to R_j

(2) Reduce R_j by eliminating tuples whose attribute A are not matching any value in R_i [A].

The cost of this semijoin is the size of projection which we denote as $s(R_i[A])$; the benefit is $s(R_j) - s(R_j')$ (suppose this semijoin reduces R_j to R_j'). If a semijoin has benefit exceeding the cost we say it is a cost-effective semijoin.

To impress that semijoin operator acts as a size reducer in distributed query processing. We compare its efficiency with IFS (Initial Feasible Solution). IFS is defined as, for a given query, all the relations involved in the query from different site are retrieved and directly shipped to the query site where the join will be executed. As a basic and simple query processing, IFS will also be use to compare against the 2 proposed algorithms in the later section. The a) and b) of figure 1 shows the cost comparison between semijoin and IFS. We can save 6 units cost during the transmission by replacing the IFS with semijoin.

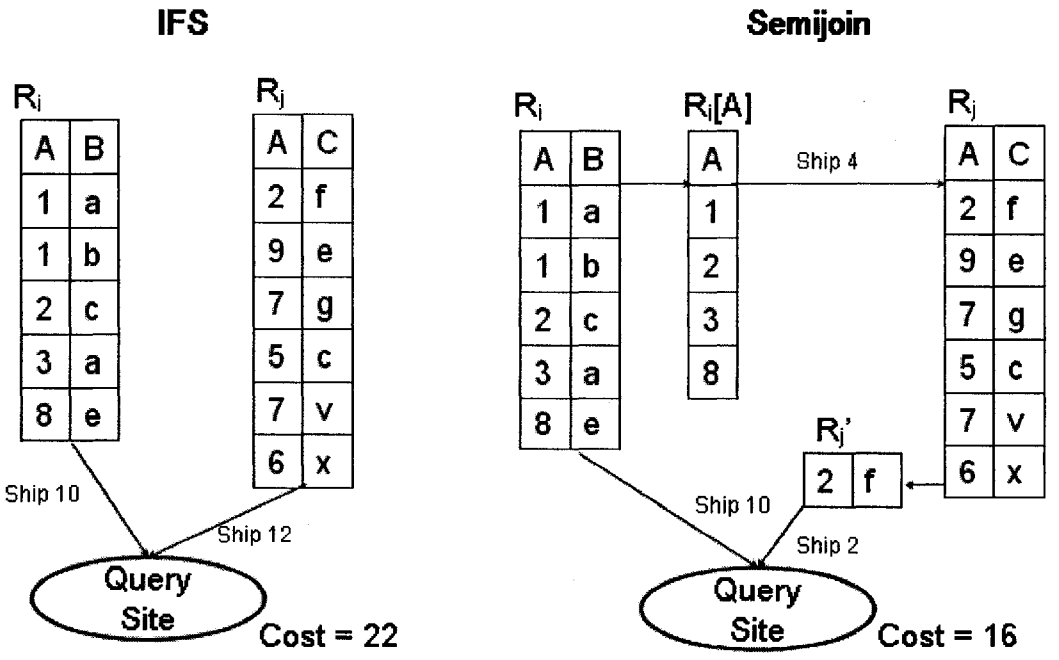

Figure 2: Example of IFS vs. Semijoin

AHY [AHY83] and SDD-1 [BC81] are two important algorithms base on semijoin. Both of them are under the assumption that at least one copy of each relation participated in the query has been chosen, then involved the reduction and assembled to the final query site.

2.4.1 The SDD-1 Query-Processing Algorithm

Algorithm SDD-1 [BC81] is the first technique based on semijoin in distributed query processing. Under the assumption of the network bandwidth being the bottleneck, the object of SDD1 Algorithm is to process the queries with a cost of inter-site data transfer as minimized as possible.

Three essential phases are involved in SDD-1 processing:
The first phase: Translate an adjective language query Q into a relational calculus form, known as an envelope, which can specify a data set as the result to Q.

The second phase: Construct relational operations called reducer P and select a site S so that the cost of compiling P and transferring results to S is the minimum between all reducers and sites. The major issue of this phase is using a greedy optimization algorithm to derive the sequence of semijoins that will retrieve the set of data needed for the query.
The third phase: Finish the local processing of Q from phase 2 at Site S. The execution in this phase uses the data from the 2nd phase and only involves local computing, therefore not discussed further in this section.

Although SDD-1 is an optimization algorithm, it still has few drawbacks and limitations. Because the selecting semijoins maximize immediate gain due to SDD-1 only pick up local optimal strategies. It will ignore the higher-cost semijoin which would result in increasing the benefits and decreasing the costs of other semijoins at each step of the strategy. Therefore, this algorithm may not be able to select the global minimum cost solution.

2.4.2 The General Algorithm (AHY)

In 1983, Apers, Hevner and Yao devised Algorithm General (Algorithm AHY) [AHY83] by improving Algorithm Serial and Parallel to deal with general query in distributed database. They figured out that it is cheaper to complete final join after relations are reduced by semijoin. They identified optimization objectives as the minimization of either response time or total execution time.

Algorithm General are constructed by following 4 phases

1) Local processing using selection and projection
2) Generate candidate relation schedules, consider each of σ join attributes a simple query
    a) Compute minimal response time for each join attribute by applying algorithm Parallel to each simple query.
    b) Compute minimal total time by applying algorithm Serial to each schedule.

3) Integrate the candidate schedules by doing procedures Response, Total and Collective.
4) Remove schedule redundancies, if necessary

There are three versions of procedures for Algorithm General. Procedure Response is used to reduce response time, while Procedure Total and Collective are used to minimize the total time.

A. Procedure Response

1) Candidate Schedule Generating and Ordering: For each relation R_i, generate candidate schedules on joining attribute d_{ij}, j = 1, 2, ..., σ. Sort these candidate schedules in ascending order of arrival time.
2) Schedule Integration: For each candidate schedule in ascending order, construct an integrated schedule for Ri that consists of the parallel transmission of candidate schedule, and select the integrated schedule with minimum response time.

B. Procedure Total

1) Generating Candidate Schedules: For each schedule containing a transmission of a join attribute d_{ij}, add another candidate schedule to minimize the total time:
2) Select the best candidate schedule: Select $BEST_{ij}$ with minimal total time of transmitting relation R_i.
3) Order the candidate schedule: $BEST_{ij}$ on joining attribute d_{ij}, j=1, 2, ... , σ, so that $ART_{i1} + C (s_1 * SLT_{i1}) \leq ... \leq ART_{i\sigma} + C (s_i * SLT_{i\sigma})$, SLT_{ij} defines the accumulated attribute selectivity of the $BEST_{ij}$ into R_i.
4) Integrate Schedules: Upon each candidate schedule $BEST_{ij}$, j=1, 2, ..., σ, an integrated schedule will be formed for relation R_i. Select the integrated schedule that results in the minimum total time value.

C. Procedure Collective

Because procedure Total does not consider the existence of redundant data transmissions in generating all candidate schedules, algorithm General may not be optimal. Procedure Collective is used to remove the most costly data transmissions.

Algorithm AHY (General) is an efficient algorithm which derives close to optimal query processing strategy. It can be applied to any general distributed query environment.

2.5 DQP strategies based on semijoin

To minimize the transmission overhead in most cost-effective way, researchers derives lots of algorithms and techniques based on semijoin to deal with variant circumstance. The following section will introduce some of them.

2.5.1 2-way semijoin

Kang and Roussopoulos proposed an extended version of semijoin which called 2-way semijoin [KR87]. A 2-way semijoin between 2 relations R_i and R_j over a attribute A can be denoted as t: $R_i \leftarrow A \rightarrow R_j$, or { s: $R_i - A \rightarrow R_j$, s^r: $R_j - A \rightarrow R_i$}. This extended semijoin is used to reduce the size of both relations R_i and R_j for the final processing in 2 directions:

Forward processing s: relation R_i is first projected on join attribute (a_{ij}), noted as R_i |A|, and is sent to R_j to reduce the size of R_j by eliminating tuples whose attribute does not match to R_i |A|.

Backward processing s^r: During the forward process, R_i |A| is divided into R_i $|A|_m$ and R_i $|A|_{nm}$. R_i $|A|_m$ is a set of values in R_i |A| that have match in R_j |A| and R_i $|A|_{nm}$ is R_i |A|- R_i $|A|_m$. R_i $|A|_m$ or R_i $|A|_{nm}$, whichever is less, will be sent back to R_i to reduce its size.

The cost of a 2-way semijoin is not as simple as C(s) + C(s^r). In most cases, the cost shall be computed as s(R_i[A]) + s*min[R_i[A], $R_i[A]_{nm}$]. C(s) + C(s^r) is valid only if s^r is delayed until s is finished, because then $R_j'[A] = R_i[A]_m$. And the benefit of a 2-way semijoin is the sum of benefits of s and s^r : [s(R_i) – s(R_i')] + [s(R_j) – s(R_j')]. Figure 2 shows how the 2-way semijoin works.

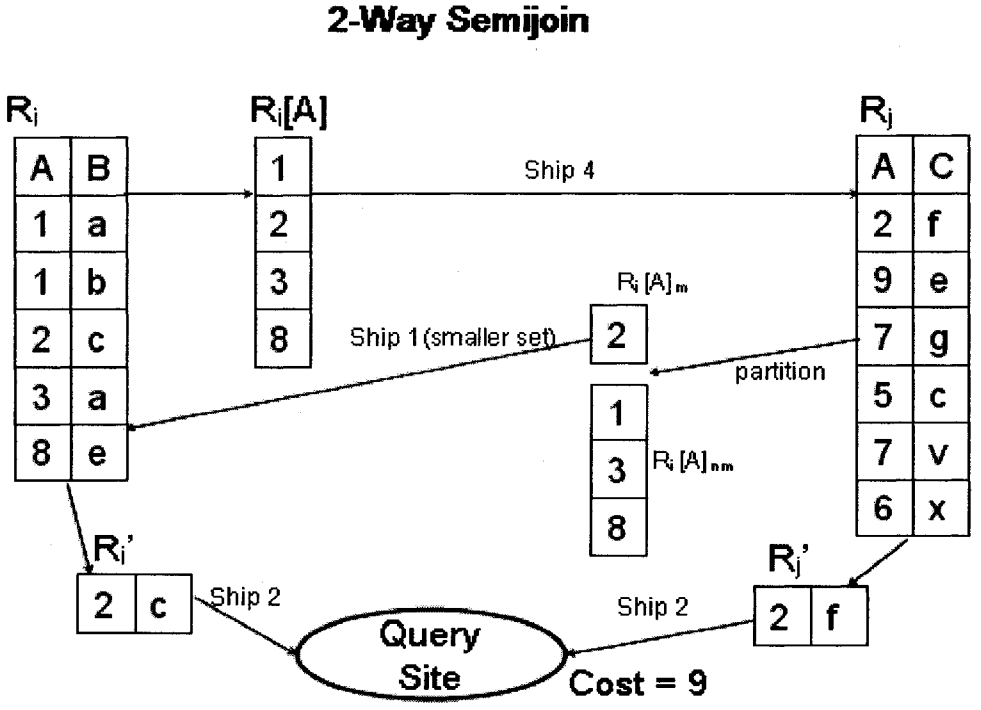

Figure 3: 2-way semijoin

It is known that an extended semijoin shall replace the tradition semijoin if it is proved more cost effective. Comparisons of reduction power effects between 2-way semijoin and traditional semijoin is given in [KR87].

When a 2-way semijoin reduce Relation R_i to R_i', Cost of this 2-way semijoin for relation R_i, is $C_{tw} = w*s_{ij} * \min[R_i |A|_m, R_i |A|_{nm}]$; Benefit of this 2-way semijoin for relation R_i is $B_{tw} = w*(|R_i| - |R_i'|)$. It is observed that $w \geq w*s_{ij}$ and $(|R_i| - |R_i'|) \geq \min[R_i |A|_m, R_i |A|_{nm}]$, therefore, net value of this 2-way semijoin $D_{tw} = B_{tw} - C_{tw} \geq 0$. A semijoin $R_i - A \rightarrow R_j$ is cost effective, while the 2 way semijoin $R_i \leftarrow A \rightarrow R_j$ is cost

effective as well. It is concluded that 2-way semijoin is always cost-effective no matter whether only one or none can be reduced cost–effectively using traditional semijoin.

Based on the study on 2-way semijoin, they developed a new join algorithm for the purpose of reducing I/O cost, a n-way pipeline algorithm. [RK91] The main goal of this pipeline algorithm is to eliminate the needs of shipping, storing, and retrieving foreign relations and(or) intermediate results on the local disks of the query site during the processing of a join, even an N-way join.

2.5.2 Composite Semijoin

Composite semijoin was proposed to deal with situations where multiple columns are involved in projection and transmission [PC90]. Typically a processing algorithm will generate numerous semijoins preformed with common source and common destination sites. However, in a situation like this, it may be of more assistance to perform semijoins as one composite against as several single sites. The authors demonstrate the possible enhancement of two classic semijoin algorithms applying composite semijoin. One is variations on Algorithm General response-time version [AHY83], using selectivity as major estimation scheme. The other is variations on Algorithm W, using "worst case elimination" as the measurement of attribute size after semijoin. Experimental results reveal that composite semijoin typically will be more beneficial against common semijoin up to 24%.

Composite semijoin is not always a superior strategy. Sometimes it may produce higher response time. Hypothetically, composite semijoin results at least as good performance if not better. This is more likely to be true if composite semijoin is replacing a parallel semijoin not a serial one. Therefore it is safe to say that it is a better approach combining semijoin and composite semijoin.

2.5.3 Domain-Specific Semijoin

In the distribute database system, many query optimization algorithms proposed for fragmented databases apply semijoins to reduce the size of the fragments of joining relations, then send the resulting to the final processing site. While the traditional semijoin can not process 2 fragments due to it may eliminate tuples before they are compared with all tuples of other joining relations. Chen and Li devise a new semijoin operator named domain-special semijoin which can be performed in a fragment-to-fragment manner [CL90]. It exploits the semantic information associated with the joining fragmented relations and provides more flexibility. As a query strategy we often use both domain-specific semijoins and semijoins. They work together can guarantee the reduction effect at least as good as the best way with only semijoin reduction.

We define the domain-specific semijoin as follows:
R_{ik} (A = B] R_{jm} = { r| r□R_i ; r.A□R_{jm} [B]□(Dom[R_i.B]–Dom[R_{jm}.B])}
Where A,B are the joining attributes, Rik and R_{jm} are two fragments of the joining relations Ri and Rj respectively. Compare with running semijoin in the horizontally partitioned database from the fig listed blow:

To estimate the size of intermediate query processing result, let R_{ik}' be R_{ik} (A = B] R_{jm}. According to the definition of domain-specific semijoin, the number of tuples reduced by R_{ik} (A = B] R_{jm} is given by:
|Rik| - | Rik'| = |Rik| (| Dom[R_{ik}.A]∩Dom[Rjm.B] | / |Dom[R_{ik}.A]|)(1 – (| R_{jm} [B]| / | Dom[R_{jm}.B] |)).
The benefit of R_{ik} (A = B] R_{jm} is Ctran(|R_{ik}| - | R_{ik}'|)w(R_i), where is the unit transmission cost and w(Ri) is the width of Ri.
The cost of R_{ik} (A = B] R_{jm} is (Ctran|Rim[B]|) w(R_j.B) .

With the estimation we can perform domain-specific semijoin with fragmented relations by following steps:

(1) Calculate its estimated benefit and cost according to the formulas presented in the previous section.
(2) If it is found to be profitable, include it in the current query-processing strategy; otherwise, ignore it.
(3) Update the related information in the database profile according to the suggested formulas in the previous section if the domain-specific semijoin is included in the current strategy.

Because domain-specific semijoin operation not only takes advantage of fragmentation design but also avoid unnecessary processing. We can get that for a given query, there is always a strategy using both domain-specific semijoins and semijoins in the fragment –fragment manner. It is at least as good as the best strategy than using semijoin reductions only.

2.5.4 Bloom Filter join (Hash Semijoin)

Tseng and Chen introduce hash semijoin as a cost saving semijoin operator in [TC92].

In a hash semijoin, also called bloom filter join, a search filter, which can be viewed as an array of bits, is transmitted between relations instead of the semijoins' projection. Initially, all bits in the array are set to 0. d hash functions hash each value in the projection into d bit addresses, setting each of the d bits to 1. Same hash function applied to the values of join attributes in apprentice relation R_j and generate another sequence of bit addresses. If all these d bits are 1 in the array, tuples in R_j containing this value will be selected as a semijoin result. $R_j \propto R_i$ is denoted as a hash-semijoin of R_i and R_j. Based on the assumption, for a specific F, the size of the bit array, the search filter is optimal when the bit array is half full of 1 bits, $F = (d/\ln 2)|R_i|$.

However, the array of the bits may not be an accurate semijoin results, because information may lose in representing a value with bits. If a bit is set to 1, it is

either the attribute actually presented or a different attribute falsely dropped due to a collision. The possibility that a value is falsely accepted by the search filter is known as a false drop, f, and the false drop probability is $f = (1/2)^d$. Then the net benefit of the hash semijoin is $BH_{ij} = (1-s_{ij}-f)w_j|R_j| - (d/\ln 2)|R_i|$. A hash semijoin program shall replace a tradition semijoin program if $BH_{ij} - BT_{ij} - fC_j > 0$, as it is more cost-effective. Figure 3 shows how the hash semijoin works.

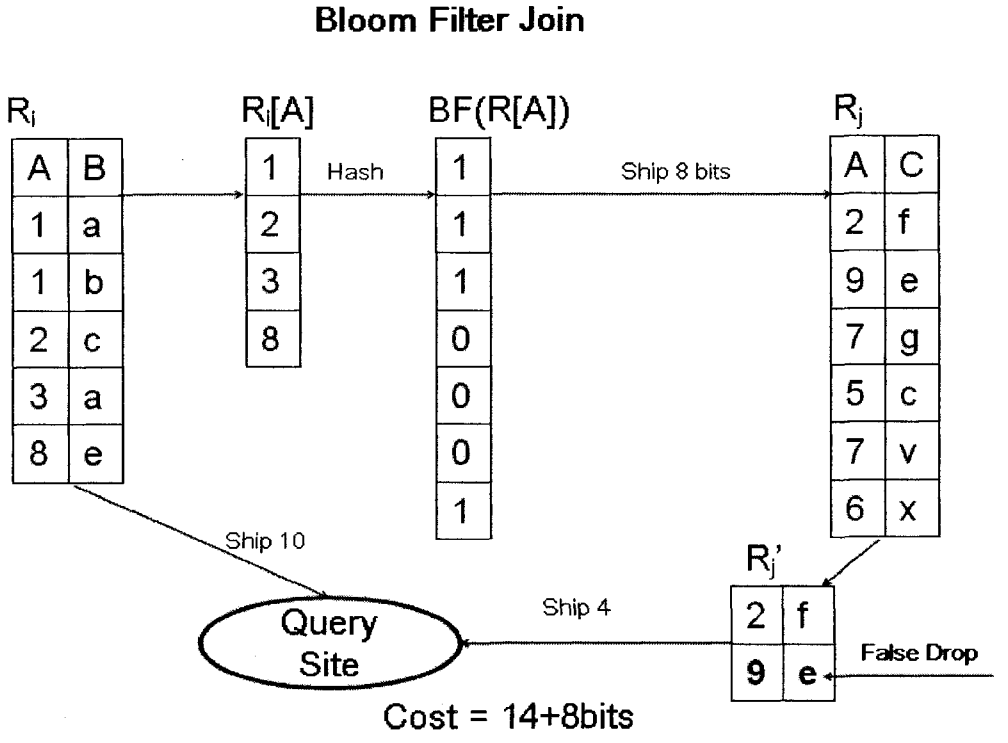

Figure 4: hash semijoin

Tseng and Chen's define that a semijoin program is more cost-effective if its summation of the potential costs of all the semijoins contained is less. The potential cost of a semijoin operation is defined as the total of the cost of executing the semijoin operation and the cost of transmitting the result of the semijoin operation. Upon such assumption, a backward replacement algorithm, to replace traditional semijoin with hash semijoin, therefore is formed. The replacement algorithm works backward through the execution tree of a

distributed query, from the leave nodes to the root node. The mechanism of this algorithm works as follows:

Step 1: Identify nodes in the execution tree, sort and mark a node by its level

Step 2: Let the lowest node be R_j, the direct predecessor of this node be R_i, $R_i - A \rightarrow R_j$.

Step 3: If $R_i \neq$ nil, replace the traditional semijoin between R_i and R_j with a hash semijoin if $CT_{ij} - CH_{ij} - fC_j > 0$ and accumulate CH_{ij} and $(s_{ij} + f)C_j$ to the potential cost of R_i, otherwise accumulate CT_{ij} and $s_{ij}C_j$

Step 4: Apply step 3 to next lowest node and end the process at the highest level.

This algorithm assumes that the semijoin program contains only traditional semijoins, which is represented by an execution tree with each node having only one direct predecessor.

Dr. Morrissey and her colleagues find that the combination of semijoins and hash-semijoin [MO99] can make better performance than use semijoin only. They also improve that the collision caused by hash function in the filter does not have a huge impact on the performance even the collision rate at 50% [MOL00].

2.5.5 PERF Join

PERF join is a novel 2-way join presented by Li and Ross [LR95]. It is designed to minimize the transmission cost during the backward phase of 2-way join.

The acronym of PERF is "Positionally Encoded Record Filters". It is based on physically tuple scan order fashion. Suppose there is a join between 2 relations R and S. the PERF is a bit vector with number of n bit (n = cardinality of relation R) which is used to represent the join information of relation S. The jth bit of the vector will set to 1 only if the jth tuple of R appears in the join result. The following figure shows the PERF for Relation R over the join with S.

R	
A	1
B	2
C	3
D	4
E	5
F	6

PERF(R)
1
0
1
0
1
0

S	
3	a
5	d
9	c
8	r
7	t
1	s

Figure 5: Example of PERF

The basic idea of PERF join is mainly based on the 2-way semijoin. In 2-way semijoin, join attribute is projected on relation R_i, and is sent to R_j to reduce the size of R_j. During the process, $R_i\ |A|_m$ or $R_i\ |A|_{nm}$, which is less, will be sent back to R_i to reduce its size. Instead of transmitting $R_i\ |A|_m$ or $R_i\ |A|_{nm}$ back to relation R_i, PERF join sends a bit vector that contains one bit for every tuple in $R_i\ |A|$. Figure 4 the principle of PERF join.

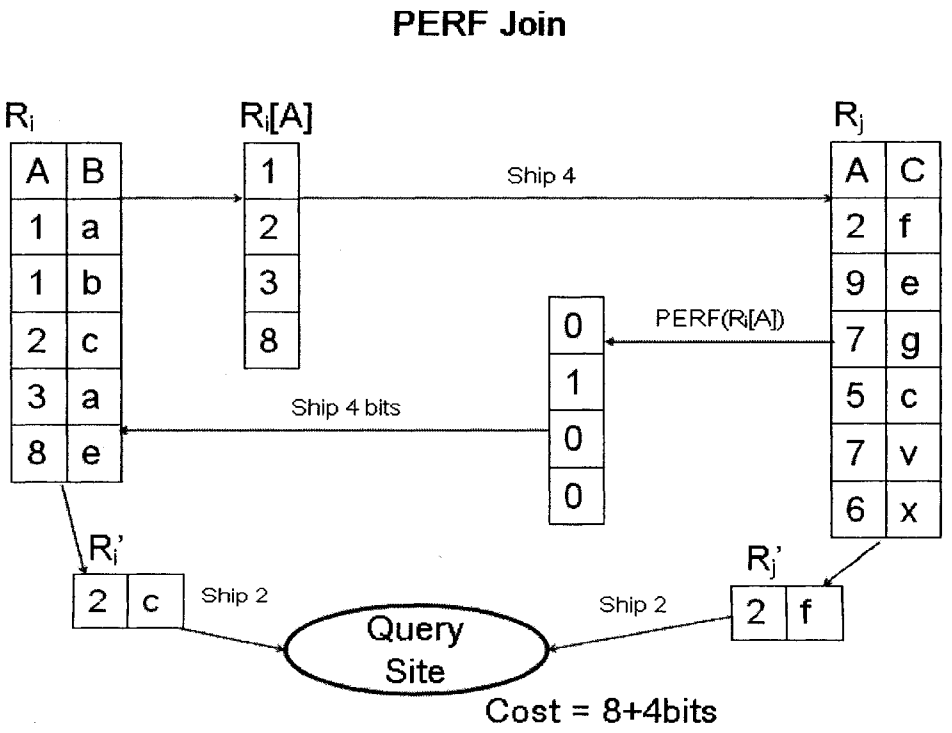

Figure 6: Example of PERF

It is known that PERF bit vector is significantly smaller in size than the 2-way semijoin result $R_i\ |A|_m$ or $R_i\ |A|_{nm}$, and at least has the same storage and transmission efficiency as a hash filter. Moreover, PERF is based on the tuple scan order, the order of bits in the bits vector which is the same tuple order of $R_i\ |A|$ that R_i initially sent. Therefore, the PERF join does not suffer any loss of join information incurred by hash collision. Another observation is that PERF join is generated after applying a real join, it shall carry complete join information and is thereby able to handle more complicated and inequality join queries, such as cyclic join queries.

Chapter 3 Implementation of Algorithms

In this chapter, we present perf join and bloom filter algorithms based on 2 way semijoin algorithm [535N] [Y2005] by replacing the join attributes projection with perf and bloom filter respectively.

3.1 Assumption

The algorithms we proposed are based on the following assumptions.

1) We assume the relational data in the Distributed Database Management System has no fragmentation or replication.
2) Only select-project-join (SPJ) query is considered. There is no set operations like UNION, INTERSECTION, PRODCT, and DIFFERENCE involved in the research.
3) A query consists of a number of relations, each of them residence at different site, and the result made available at the query site. Each relation can have a number of join attributes
4) We assume the cost model is $C(X) = C_0 + X$, where $C_0 = 0$ for simplicity. X is the amount of data transmitted.
5) We assume that we have a perfect hash function which the filter size is the same as the domain size.
6) When a semijoin operation occurred between 2 relations. Only the currently involved join attributed size will be reduced while other attributes properties will has no change.

3.2 Algorithms Description

The basic idea of our proposed algorithm is use bloom filter and perf filter to instead of the original semijoin projection during the procedure of data transmission.

3.2.1 Algorithm Bloom Filter

In this algorithm, we will apply bloom filter both in forward reduction phase and backward reduction phase. In the forward reduction phase, we construct the bloom filter for join attribute projection. While in the backward reduction phase, we need to build a bloom filter for the join match set. The detail steps of the algorithm are listed below.

Steps of Algorithm BF:

1. Arrange all relations by size in ascending order such that $S(R_i) \leq S(R_j) \leq S(R_k) \leq \ldots \leq R_m$.

2. For each joining attribute, we get a list of relations which contain this attribute by the order of step1.

3. Generate an execution schedule $R_i \rightarrow R_j \rightarrow R_k \ldots R_m \ldots R_k \rightarrow R_j \rightarrow R_i$ for each of joining attribute. $R_i \rightarrow R_j \rightarrow R_k \ldots \rightarrow R_m$ is forward reduction while $R_m \ldots \rightarrow R_k \rightarrow R_j \rightarrow R_i$ we call backward reduction.

4. Start to execute schedules. The schedule with the smallest first projection size will be executed first.

5. For each predicate between the 2 relations in forward reduction phase of the schedule. We generate the bloom filter which has the same bit number with domain size base on our assumption. If the bloom filter size is smaller than projection size and net benefit is greater than 0, we execute this

bloom filter join by sending the bloom filter form R_i to R_j. Otherwise we ignore it and execute the bloom filter join between next 2 relations over the join attribute. For example, if the net benefit of $R_i \rightarrow R_j \leq 0$ or $BF(R_i[A]) > |R_i[A]|$, we ignore this and execute $R_i \rightarrow R_k$

6. Once a semijoin has been done in forward reduction phase. We need to record it in an arraylist which we called semijoin list. And also keep the join match set and non match set in a data structure.

7. Start backward reduction follow the schedule. For relation R_j, R_i over join attribute A, if it exists in the in bloom filter execution list, it means the bloom filter join from R_i to R_j on attribute A has been done already in the forward reduction phase, thus, we could find out the match set (m_set) or non match set (nm_set) for $R_i[A]$ of relation R_j and generate another bloom filter for either match set or none match set due to both of them will have the same size in bloom filter. Compare the size of bloom filter for the match set and min {m_set, nm_set}. send the smaller one back to R_i to reduce R_i. The transmission cost is the size of the smaller one which has been sent. If there is no record of relation R_j, R_i over join attribute in the executed bloom filter join list which means there was no bloom filter join executed from R_i to R_j on attribute A. so we need to consider to do the bloom filter by the term of net benefit >0 and also the bloom filter size is smaller than projection size.

8. Repeat steps 4,5,6,7 till all of schedules have been processed.

9. Output the transmission cost which is the sum of every reduced relation size plus the transmission cost in every single schedules.

3.2.2 Algorithm PERF

Compare with 2-way Semijoin, PERF join is has better performance if and only if $1<W(A) * \min(\rho(R), 1-\rho(R))$. It means that PERF join can not exceed 2-way semijoin all the time. There still exists chance for 2-way semijoin could have less transmission cost than PERF join when $\min(\rho(R), 1-\rho(R))$ is very low. To make better efficiency, we apply one of them which have lower cost into the back reduction phase of 2-way semijoin algorithm. Therefore the new algorithm could perform at least as the old one. The core of this algorithm is based on Algorithm UPSJ [Yang05]. The detail steps of the algorithm are listed below.

Steps of Algorithm PERF:

1. Arrange all relations by size in ascending order such that $S(R_i) \leq S(R_j) \leq S(R_k) \leq \ldots \leq R_m$.

2. For each joining attribute, we get a list of relations which contain this attribute follow the order from step 1.

3. Generate an execution schedule $R_i \rightarrow R_j \rightarrow R_k \ldots R_m \ldots R_k \rightarrow R_j \rightarrow R_i$ for each of joining attribute. $R_i \rightarrow R_j \rightarrow R_k \ldots \rightarrow R_m$ is forward reduction while $R_m \ldots \rightarrow R_k \rightarrow R_j \rightarrow R_i$ is backward reduction.

4. Start to execute schedules. The schedule with the smallest first projection size will be executed first.

5. For each predicate between the 2 relations in forward reduction phase of the schedule, only execute the semijoin by the term if the net benefit is greater than 0. Otherwise execute the next predicate. For example, if the net benefit of $R_i \rightarrow R_j \leq 0$, we ignore this and execute $R_i \rightarrow R_k$

6. Once a semijoin has been done in forward reduction phase. We need to record it in an arraylist which we called semijoin list. And also keep the join match set and non match set in a data structure.

7. Start backward reduction follow the schedule. For a semijoin between relation R_j, R_i over join attribute A, if it exists in the semijoin list, it means the semijoin from R_i to R_j on attribute A has been done already during the forward reduction phase. Thus, we could generate the bit vector PERF(R_i) for the projection of R_i[A]. while we could also find out the match set(m_set) and non match set(nm_set) for R_i[A] of relation R_j. compare the size of PERF(R_i) and min(m_set, nm_set), send the smaller one back to Ri to reduce R_i. The transmission cost is the size of the smaller one which has been sent. If the semijoin between relation R_j, R_i over join attribute A does not exist in the executed semijoin list which means there was no semijoin happened from R_i to R_j on attribute A. so we need to consider to do the semijoin or not by the term of net benefit >0. If net benefit > 0, we do this semijoin, the cost is $|R_j[A]|$ * width of A. otherwise we leave 2 relations without executing semijoin.
8. Repeat steps 4,5,6,7 till all of schedules have been processed.

9. Output the transmission cost which is the sum of every reduced relation size plus the transmission cost in every single schedules.

3.2.3 Example of Proposed Algorithms

The following example will show how the algorithm Perf join and Bloom Filter join work. Suppose we have a query "List the P#, PNAME and total quantity for all parts that are current on order from suppliers who supply that part to jobs." In this query, there are two joining attribute which are P# and S#. Assume that each relation is located at different network node. After the local processing, the query can be represented as the SQL listed below:

"select * from PARTS, ORDER, SPJ where PARTS.P#=ON-ORDER.P#=SPJ.P# and ON-ORDER.S#=SPJ.S#."

We can use figure7 which is listed above to represents the query data. In the table, the size of relation is denoted as S_i, the selectivity and the size for each individual are represented by b_{ij} and ρ_{ij}. We set join attribute domain size for the P# is 1000 while domain size for the S# is 500 here.

Relation	Size S_i	P#		S#	
		b_{i1}	ρ_{i1}	b_{i2}	ρ_{i2}
R_1: On-Order	1000	400	0.4	100	0.2
R_2:S-P-J	2000	400	0.4	450	0.9
R_3:Parts	3000	900	0.9	-	-

Figure 7: Table of a Query

Example of Algorithm BF

Step1: There are 2 joining attributes in the query, for each joining attribute, get a list of the relations which have that attribute.

P#: R_1, R_2, R_3

S#: R_1, R_2

Step2: Order the relations by ascending size.

$R_1 < R_2 < R_3$

Step3: Construct execution schedule for each join attribute.

P#: $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_2 \rightarrow R_1$

S#: $R_1 \rightarrow R_2 \rightarrow R_1$

We separate each of schedules to two phase which are forward reduction phase and backward reduction phase. For example, in schedule of P#, $R_1 \rightarrow R_2 \rightarrow R_3$ is forward reduction phase while $R_3 \rightarrow R_2 \rightarrow R_1$ is backward reduction phase.

Step4: Order the schedules by ascending size of join attribute projection of the first relation of each schedule.

S#: $R_1 \rightarrow R_2 \rightarrow R_1$ $R_1[S\#] = 100$

P#: $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_2 \rightarrow R_1$ $R_1[P\#] = 400$

Step5: Start to execute the schedule for S#: $R_1 \rightarrow R_2 \rightarrow R_1$.

1) Execute the forward reduction phase $R_1 \rightarrow R_2$:

   We need to decide the bloom filter join form R1 to R_2 over attribute S# should be executed or not. The BF(R_1 [S#]) = domain size/8 =500/8= 63 bytes which is smaller than the projection size of S#. The benefit is 1600 which derived by size of R_2*(1- ρ (R_1 [S#])). The Net Benefit = Benefit – Cost=1600-63>0, so we need to do this bloom filter join. After execute this bloom filter join, Relation R_2 has been reduced to R_2' with the size 400 derived from 2000 * 0.2. The attribute size and selectivity of joining attribute has been updated to 90, and 0.18 respectively. We record this bloom filter join in the bloom filter join list.

2) Execute the backward reduction phase $R_2 \rightarrow R_1$:

   First we search the bloom filter join list to see the bloom filter join between R_1 and R_2 over the attribute S# exists or not. In this case, it is in, which means that the bloom filter join from R_1 to R_2 has been done. We can generate bloom filter for the match set, which is also 500/8=63. For here, because the size of match set m_set= 20(100*0.2) is smaller, Then we just need to send the match set back to the relation R_1. And the relation R_1 will be reduced to R_1' with the size 180.

After that, there is no more relation to be considered in the schedule S#. The transmission cost of schedule S# is 63+20 bytes, the size of relation R_1 and R_2 has been reduced to 180 and 400 respectively. The execution schedule for S# has been done.

We start to process next schedule for P# with the same process with schedule S#. Our algorithm keeps running until there are no more schedules left. After that, all reduced the relations will be sent to the final query site to participate the global query. The output of our algorithm is sum of all reduced the relations' size plus each sum of the cost occurred in the every single schedule.

Example of Algorithm PERF:

For the perf join, the process from step1 to step 4 will be same as the Algorithm BF. It will generate same execution schedules for the query listed below:

S#: $R_1 \rightarrow R_2 \rightarrow R_1$

P#: $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_2 \rightarrow R_1$

Step5: Start to execute the schedule for S#: $R_1 \rightarrow R_2 \rightarrow R_1$.

1) Execute the forward reduction phase $R_1 \rightarrow R_2$:
   We need to decide the semijoin form R_1 to R_2 over attribute S# should be executed or not. The cost is 100. Benefit is 1600 which derived by size of R_2*(1- ρ (R1[S#])). The Net Benefit = Benefit – Cost=1600-100=1500>0, so we need to do this semijoin. After execute this semijoin, Relation R_2 has been reduced to R_2'. The size of R_2' is 400(2000 * 0.2) now. The attribute size and selectivity of joining attribute has been updated to 90, 0.18 respectively. We record this semijoin in the semijoin list.

2) Execute the backward reduction phase $R_2 \rightarrow R_1$:
   First we search the semijoin list to see the semijoin between R_1 and R_2 over the attribute S# exists or not. In this case, it is in, which means that the semijoin from R_1 to R_2 has been done and site R_2 already has the join attribute projection information of R_1[S#]. It can generate PERF (R_1[S#]) which size is |d|/8 = 100/8=13 bytes. The size of match set m_set=20 (100*0.2), and the size of none match set nm_set = 80. Comparing the size between PERF (R_1 [S#]) and min (m_set, nm_set), we find the smaller one is the PERF (R_1). Then we can do PERF join on back

reduction phase. The cost is the size of PERF which is 13 and the relation R_1 will be reduced to R_1' with the size 180.

After that, there is no more relation to be considered in the schedule S#. The cost of schedule S# is 100+13 bytes, the size of relation R_1 and R_2 has been reduced to 180 and 400 respectively.

We start to process next schedule for P# with the same process with schedule S#. Our algorithm keeps running until there are no more schedules left. After that, all reduced the relations will be sent to the final query site to participate the global query. The output of our algorithm is sum of all reduced the relations' size plus each sum of the cost occurred in the every single schedule.

Chapter 4 Experiment and Evaluation

To evaluate the actual performance of the proposed algorithms, we carried out multitudinous experiments based on various scenarios and queries. In this chapter, we describe our methodology, present detail of our experiments, and discuss the final result.

4.1 Methodology

The platform for evaluating the proposed algorithms has to achieve the following objectives.

1. To measure the performance enhancement of Algorithm PERF and Algorithm BF over the IFS respectively.
2. To compare the performance of Algorithm PERF and Algorithm BF under a wide variety of distributed queries.

The following formulas are used to calculate the performance enhancement between the algorithms:

IFS vs. Proposed algorithm:

$$\frac{\text{Cost(IFS)-Cost(Proposed Algorithm)}}{\text{Cost(IFS)}} \text{ *100\%=Percentage Improved}$$

Algorithm 1 vs. Algorithm 2:

$$\frac{\text{Cost(Algorithm 1)-Cost(Algorithm 2)}}{\text{Cost(Algorithm 1)}} \text{ *100\%=Percentage Improved}$$

From experiment, we need to find out the answers for the following involved questions.

1. How does the number of the relations in the query affect the performance?
2. How does the number of attributes in the query affect the performance?
3. How does the selectivity of the attributes affect the performance?
4. How does the domain size affect the performance?

4.2 Test Query and Platform

The experiment system involves query generator, proposed algorithms and the analysis program. Some details will be given later in this section.

4.2.1 Test Query (query generator)

The experiments system takes large amount of queries as input to evaluate proposed algorithms. The form of a query we have already represented in Figure 8. It contains the following characteristics:

- Number of relations: Each query consists of arbitrary number of relations from 3-6.

- Number of join attributes: Each relation have arbitrary number of join attributes from 2-4.

- Relation cardinality: the number of tuples or records in a relation. Each relation in the query has between 500 and 4000 tuples.

- Attributes domain size: the total number of distinct attribute values an attributed can possible contain. We fix all join attribute domain size to 1000 in our experiment system.

- Selectivity: the ratio of distinct attribute values out of the number of all possible values of a join attribute. Suppose the cardinality of the joining

attribute is |d|, the domain of the d is Dom, p=|d|/Dom (d). Generally, the selectivity of an attribute is an estimate of the ability of the attribute to reduce the size of the relations. A joining attribute has high selectivity if the ratio is low while low ratio denotes the high selectivity. For example, a selectivity of 0.1 is considered high while a selectivity of 0.9 is low.

- Query Type: In our experiment platform, the relations is picked from 3 to 6, the arbitrary number of joining attributes is form 2-4. Selectivity is divided to 3 level which are high (0.1-0.4), med (0.4-0.7), low (0.7-0.9). We use these 3 factors to name a query type. A type of 6-3-0 represents a query which has 6 relations, 3 joining attributes and the selectivity range of the attribute is from 0.1 to 0.4. With this rule, we will have 36 different kind of queries total. For each kind of query type, we will generate 10 queries with random data, and get the average output as final result

We save these kind of information as XML format file. The example fragment of XML file listed below in Figure 8 denotes one query. 360 pieces of fragment like this one constructs the input queries file of our propose algorithms.

```
<?xml version='1.0' ?>
<query1 type="3-2-0" domain = "1000">
        <relation name="R1" size="1000">
                <attribute name="P#" selectivity="0.4" size="400" />
                <attribute name="S#" selectivity="0.2" size="100" />
        </relation>
        <relation name="R2" size="2000">
                <attribute name="P#" selectivity="0.4" size="400" />
                <attribute name="S#" selectivity="0.1" size="100" />
        </relation>
        <relation name="R3" size="3000">
                <attribute name="P#" selectivity="0.1" size="100" />
        </relation>
</query1>
```

Figure 8: Example XML of a Query

4.2.2 Platform Implementation

The experiment platform is developed with Microsoft C# based on object-oriented concept. Several classes as show in the following figure are constructed in this program to represent the data structures used in the algorithm. The hierarchy of basic class tree is illustrated in Figure 9.

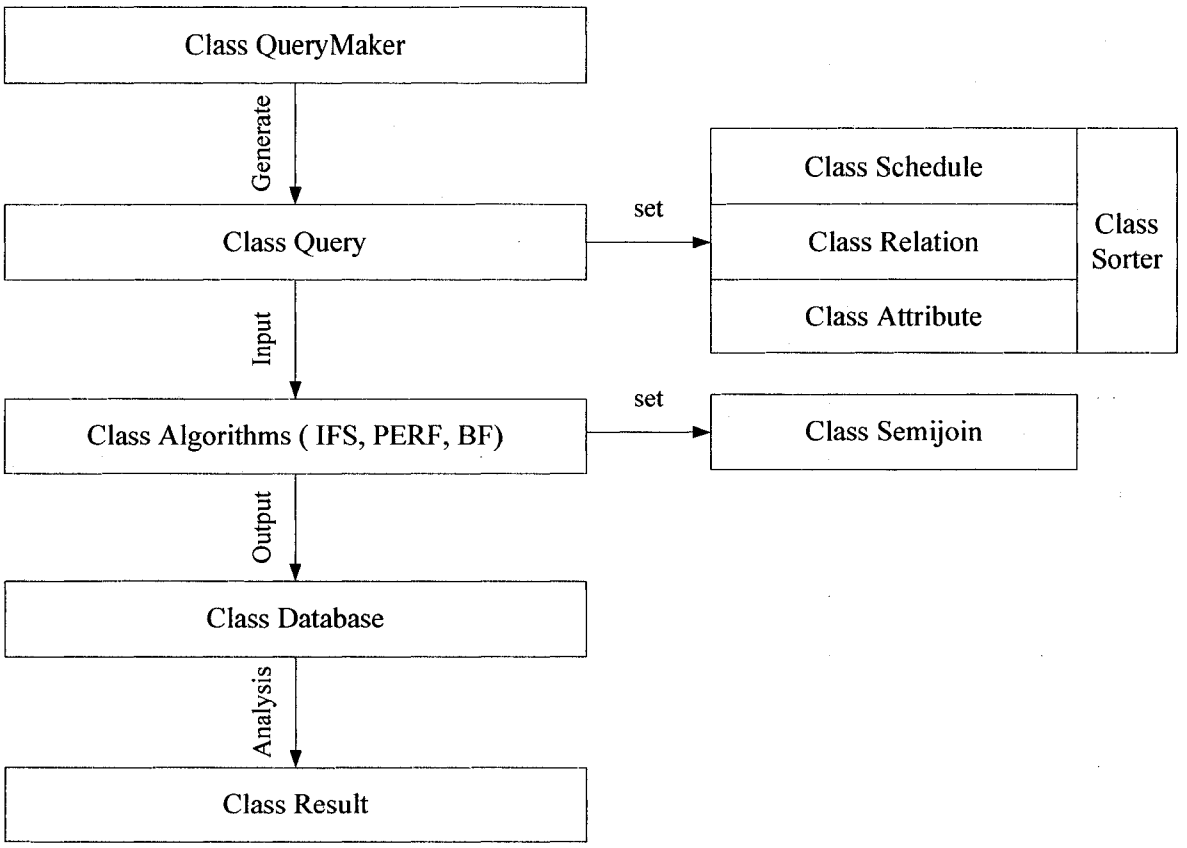

Figure 9: Hierarchy of classes.

The descriptions of above classes are as follows:

- ***Query Maker:*** for each kind of query type (etc 6-4-0), generate 10 queries' xml nodes. Save all these nodes to one xml file. the detail rule of this process has been introduced in 4.2.1

- ***Query:*** For each query node in the read in xml file, we generate a query. Each query has an arraylist to save the schedules of the query. The process of the query generating is also the process of initialization for schedule, relation, and attribute.

- ***Schedule:*** Represent the sequence of the relation execution. Each schedule instance contains a list of relation ordered by the size of the

projection of the attribute it refers to. It also has an array list to mark the semijoin which has been executed by which 2 relations. The cost used to record how many bytes has been transferred during this schedule.

- ***Relation:*** Represent the relation's information like name, size and contained attributes. Attributes list use to keep the joining attributes information of this relation.

- ***Attribute:*** Represent single joining attribute information in a relation such as cardinality, name, selectivity and size.

- ***Sorter:*** use to sort the size or cost etc....

- ***Semijoin:*** represent a semijoin was executed by which 2 relations over the joining attributed. In the backward reduction phase, we need to decide whether we should apply the proposed join to the schedule by the term of checking a semijoin has been done between these 2 relation during the forward reduction phase or not.

- ***Algorithms:*** take a query as input, run it with algorithm IFS, HASH, PERF respectively, and output is the total cost for each of algorithm. The total cost is the summation of the transmission total cost and total reduced relation size.

- ***Database:*** once an instance of a single query has been done by the algorithms. We will insert the query result data into the database.

- ***Result:*** use to analysis and display the experiment data.

4.3 Result Evaluation

After a 10 queries for 36 types of queries as input to our proposed algorithms, we have the experiment result data listed below:

Type	IFS	BF	IMP(BF)	PERF	IMP(PERF)	BF/PERF
3-2-0	3854	758	80.33%	1100	71.46%	45.12%
3-2-1	7388	2190	70.36%	3864	47.70%	76.44%
3-2-2	7152	4838	32.35%	6056	15.32%	25.18%
3-3-0	5848	834	85.74%	1110	81.02%	33.09%
3-3-1	8630	2518	70.82%	4380	49.25%	73.95%
3-3-2	12696	5762	54.62%	10162	19.96%	76.36%
3-4-0	9588	1132	88.19%	1550	83.83%	36.93%
3-4-1	15052	2800	81.40%	5260	65.05%	87.86%
3-4-2	12494	5446	56.41%	7730	38.13%	41.94%
4-2-0	7610	826	89.15%	1176	84.55%	42.37%
4-2-1	9392	2240	76.15%	3748	60.09%	67.32%
4-2-2	10124	4208	58.44%	6966	31.19%	65.54%
4-3-0	10776	1134	89.48%	1544	85.67%	36.16%
4-3-1	13910	2820	79.73%	5242	62.31%	85.89%
4-3-2	13874	5434	60.83%	8190	40.97%	50.72%
4-4-0	15582	1556	90.01%	2140	86.27%	37.53%
4-4-1	14510	2880	80.15%	5614	61.31%	94.93%
4-4-2	18976	5888	68.97%	14692	22.58%	149.52%
5-2-0	11120	1060	90.47%	1448	86.98%	36.60%
5-2-1	11934	2528	78.82%	4174	65.02%	65.11%
5-2-2	13766	4356	68.36%	9666	29.78%	121.90%
5-3-0	13414	1426	89.37%	1986	85.19%	39.27%
5-3-1	14960	2792	81.34%	5066	66.14%	81.45%
5-3-2	17238	4708	72.69%	10672	38.09%	126.68%
5-4-0	14876	1502	89.90%	2014	86.46%	34.09%
5-4-1	16694	2876	82.77%	4870	70.83%	69.33%
5-4-2	24092	5878	75.60%	13456	44.15%	128.92%
6-2-0	11226	918	91.82%	1160	89.67%	26.36%
6-2-1	14572	2516	82.73%	4106	71.82%	63.20%
6-2-2	17206	4826	71.95%	11764	31.63%	143.76%
6-3-0	13182	1094	91.70%	1430	89.15%	30.71%
6-3-1	20586	3564	82.69%	5820	71.73%	63.30%
6-3-2	21894	5304	75.77%	14544	33.57%	174.21%
6-4-0	16192	1630	89.93%	2198	86.43%	34.85%
6-4-1	27624	4708	82.96%	8458	69.38%	79.65%
6-4-2	26086	5974	77.10%	14806	43.24%	147.84%
AVG	**14003**	**3081**	**78.00%**	**5782**	**58.71%**	**87.66%**

Figure 10: Experiment Result of Algorithm IFS, BF and PERF

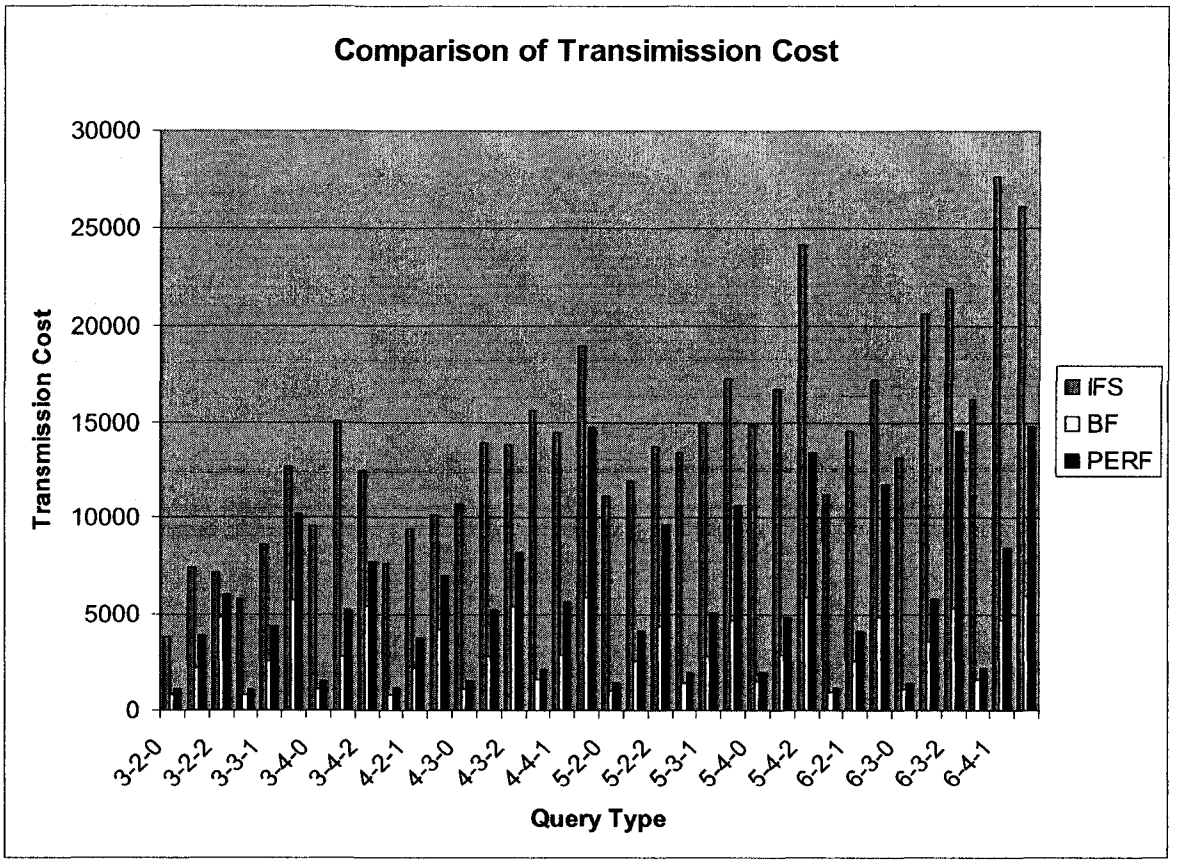

Figure 11: Transmission Cost Comparison of IFS, BF and PERF

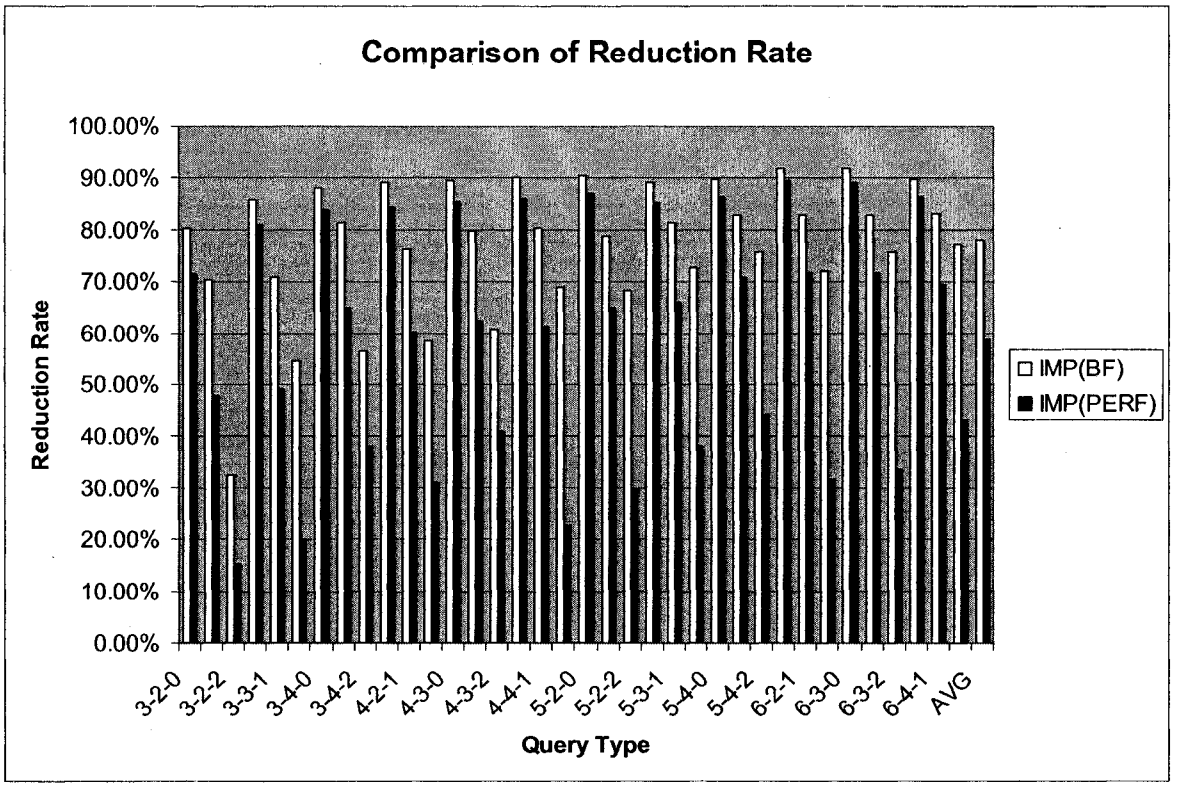

Figure 12: Reduction Rate Comparison between BF and PERF

Figure 11 and 12 show the transmission cost comparison of BF, PERF with IFS and transmission reduction rate comparison of propose algorithms over the IFS. Our propose algorithms make greater improvement on performance than IFS. Algorithm BF exceeds the IFS over 91.82% at most while Algorithm PERF can make at most 89.67% improvement.

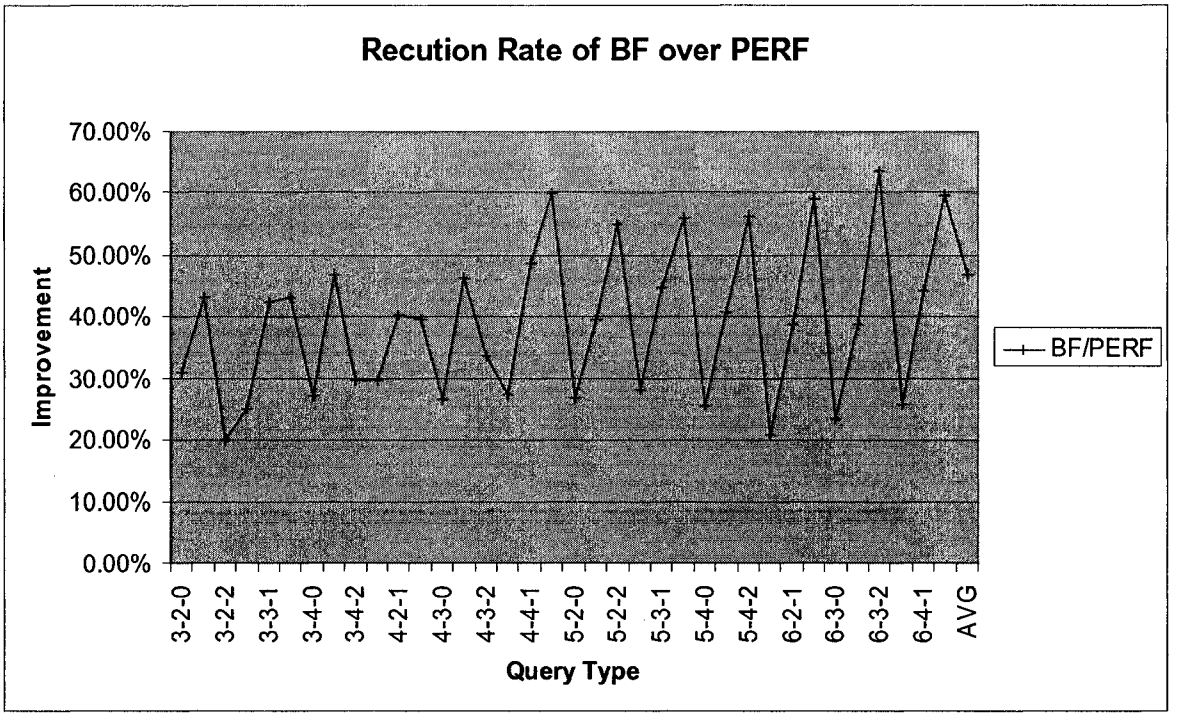

Figure 13: Reduction Rate of PERF over BF

Figure 13 shows the improvement percentage of Algorithm BF over Algorithm PERF derived by the formula listed in 4.1. As it is showed in the above chart, the peak values represent the different between 2 algorithms. The larger value the query type is, the more transmission cost can be reduced than the algorithm PERF produced. We also noticed that with more join attributes and relations join into the query, the different of performance between two algorithms produced almost keep the same level in the low selectivity range. But the algorithm BF produces perceptible enhanced performance with a higher selectivity.

Conclusion can be driven from the above figures that comparing to IFS, both algorithm BF and PERF are able to produce a high reduction power over distributed query transmission cost. The average results for both algorithm shows that the BF is over-performed than PERF. However, to find out and answer for the questions we listed in 4.1. We need to do further investigation on other factors affecting query transmission, such as selectivity, number of relation and join attributes, domain size etc...

4.3.1 Effects of the Selectivity Level

In this scenario, we will find how the cost reduction rate of the proposed algorithms will be related by selectivity. As we know, selectivity is defined as a ratio of distinct attribute values over the attribute domain size. As the most important factor in a distributed query, it can be use to estimate the reduced size of a join attribute during a semijoin operation. The selectivity range in the query is between 0.1 and 0.9.

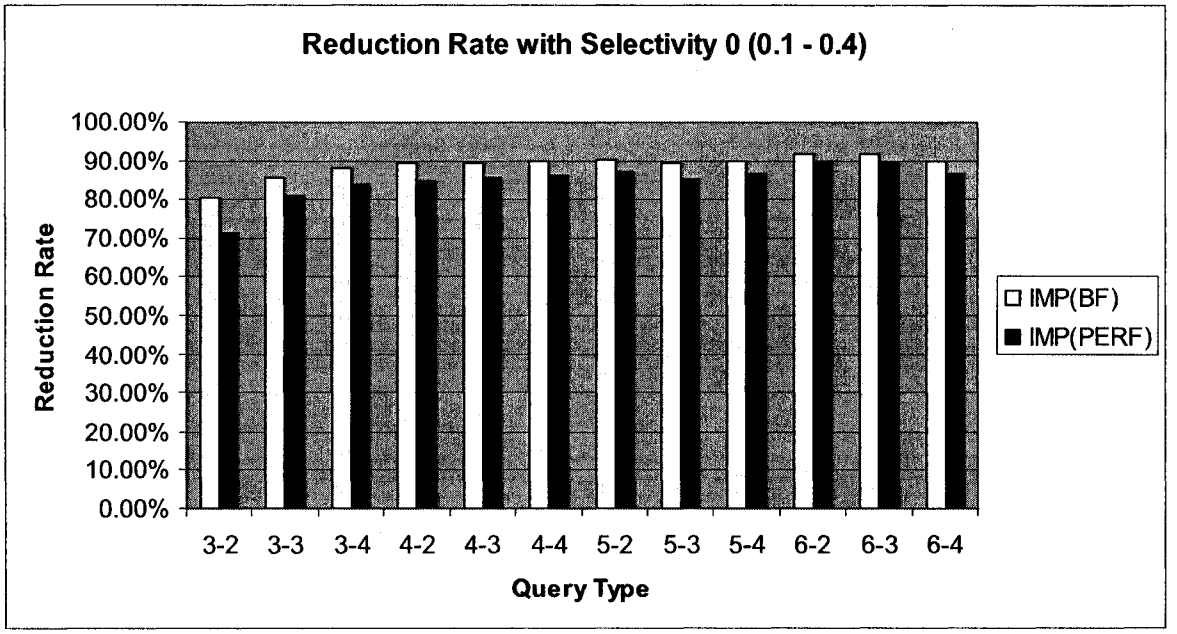

Figure 14: Reduction Rate with High Selectivity Level (0)

Figure 14 illustrates reduction rate comparison with the low selectivity level for both algorithms. BF improves the reduction rate from a low percentage of 80.3% to 91.82%, while Algorithm PERF does from 71.5% to 89.7%.

Experimental data shows that, both Algorithm BF and Algorithm PERF show the great performance in high selectivity level. The more relations and join attribute a query has, the better reduction rate that both algorithms produce. However, with the distributed query of high selectivity, the performance of Algorithm BF outperforms than Algorithm PERF.

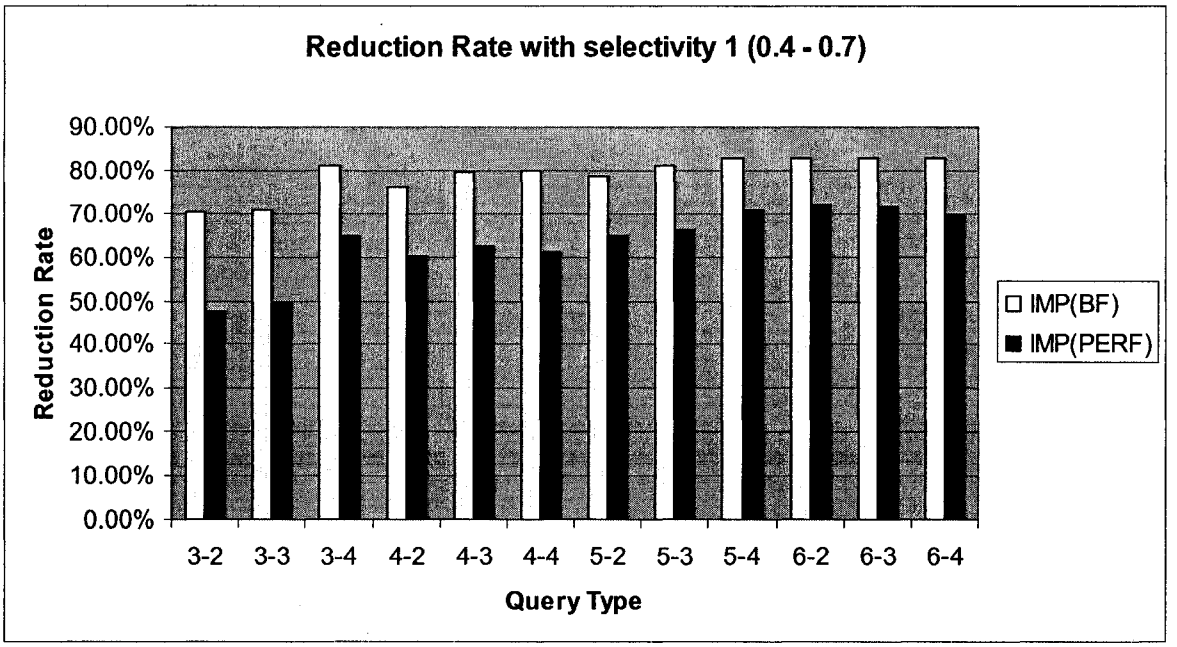

Figure 15: Reduction Rate with Medium Selectivity Level (1)

As it shows in Figure 15, with medium selectivity, the lowest reduction rate that the proposed algorithms can produce is at least 67.7% by Algorithm BF as it is 58.5% by Algorithm PERF. The highest value of improved reduction rate from BF is 85.7% as it is 86.9% by PERF. The reduction rate begins to descend as selectivity grows.

At a selectivity range of 0.4-0.7, the performance both algorithms produced is still competitive and keeping growing with an increasing relation and attribute number. Rather remarkable, BF over-performed Perf at the beginning but failed to keep the advanced position later on after relation and attribute reach certain numbers.

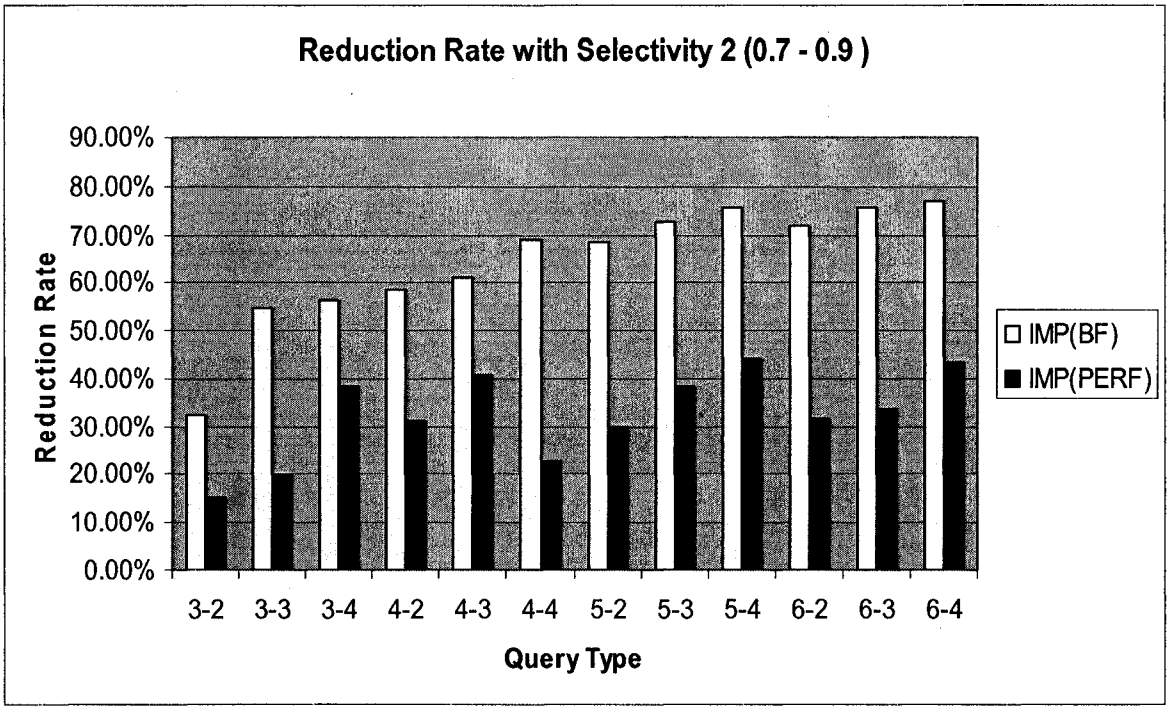

Figure 16: Reduction Rate with Low Selectivity Level (2)

As it shows in Figure 16, with low selectivity, the lowest reduction rate that the proposed algorithms can produce is at least 32.4% by Algorithm BF as it is 15.3% by Algorithm PERF. The highest value of improved reduction rate from BF is 77.1% as it is 44.2% by PERF. Reduction rate reaches its lowest value while selectivity decreases its own range.

When the selectivity is very low, both Algorithm BF and Algorithm PERF's performance decrease obviously. Especially for PERF, the reduction rate is down to around 15% under low relation and attribute. Yet, performance of both algorithms bounced back with the growth of relations and attributes.

4.3.2 Effects of the Number of Relations

In this section, we will find how the cost reduction rate of the proposed algorithms will be related by the number of the relations.

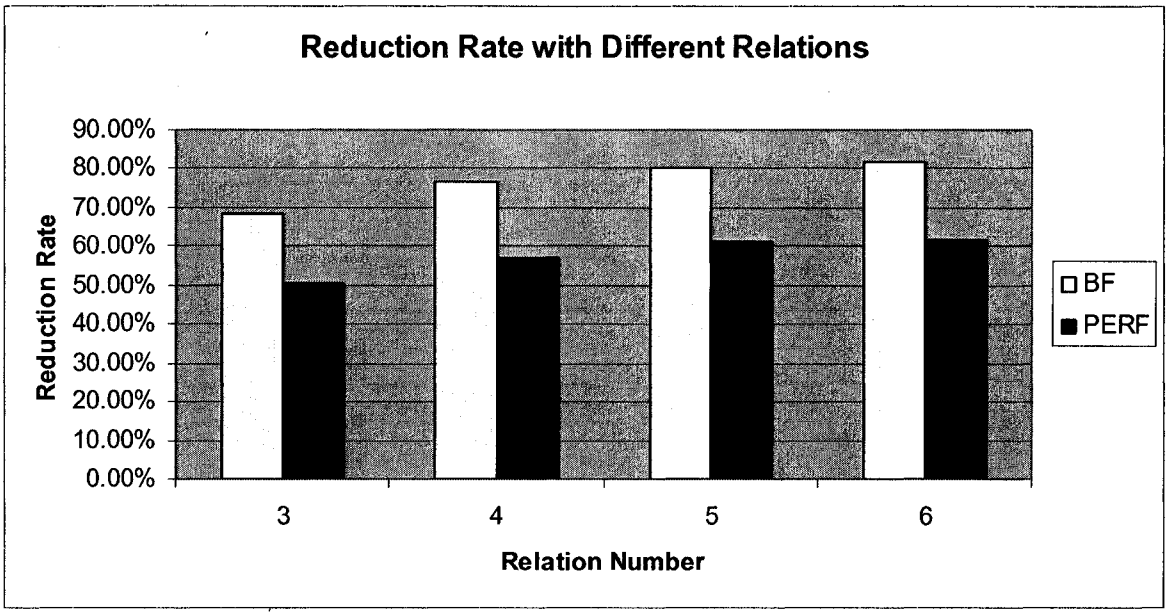

Figure 17: Reduction Rate with Different Relation

In figure 17, with the queries of relation range from 3 to 6, the average of reduction cost that bloom filter join can produce better than perf does. However, with more relation joining to the query, the performance of both algorithms is getting better.

4.3.3 Effects of the Number of Attributes

In this section, we will examine how the number of join attributes will affect the cost reduction rate of the proposed algorithms.

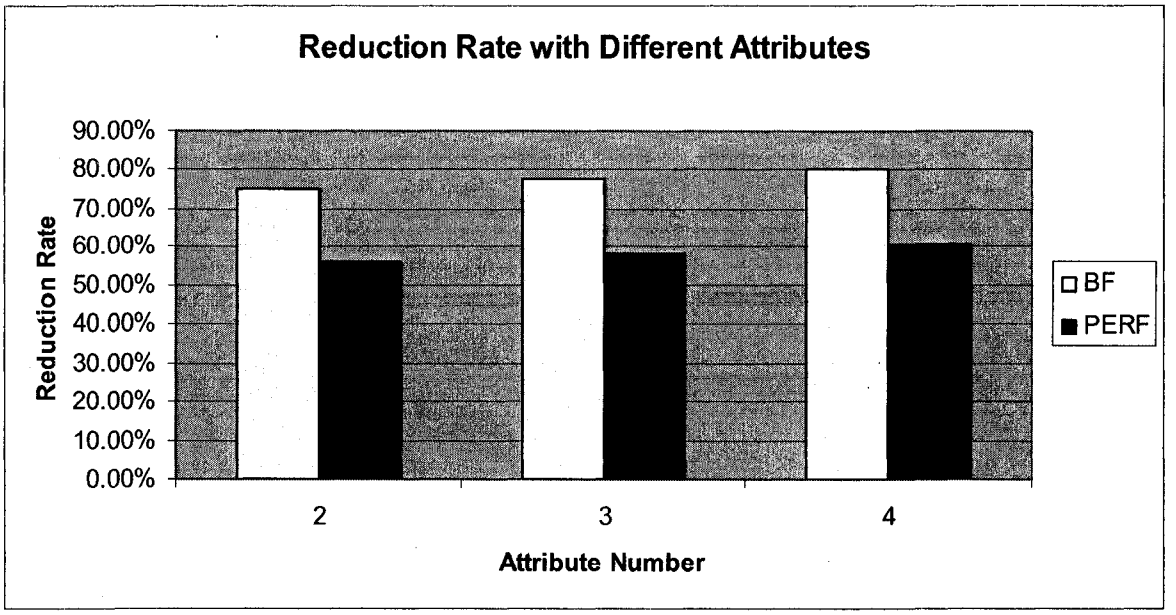

Figure 18: Reduction Rate with Different Attribute

In figure 18, from 2 attribute to 4 attributes the average of reduction cost of bloom filter join still better than perf. However, under the condition of involving more join attributes, a more competitive reduce rate will be demonstrated.

4.3.4 Effects of the Domain Size

In this scenario, we are looking forward to present a methodology on measuring that how the domain size will influence the performance of our proposed algorithms, in a more self-revealing way. We pick up a group of queries with 3 relations, 2 join attributes and medium selectivity level (3-2-1) and try to increase their domain size from 1000 to 2000. Due to the fact that domain size is a responsive parameter of our algorithms – it produces a direct effect on bloom filter size. Since domain variable will cause the change to cardinality, and therefore shape PERF size accordingly.

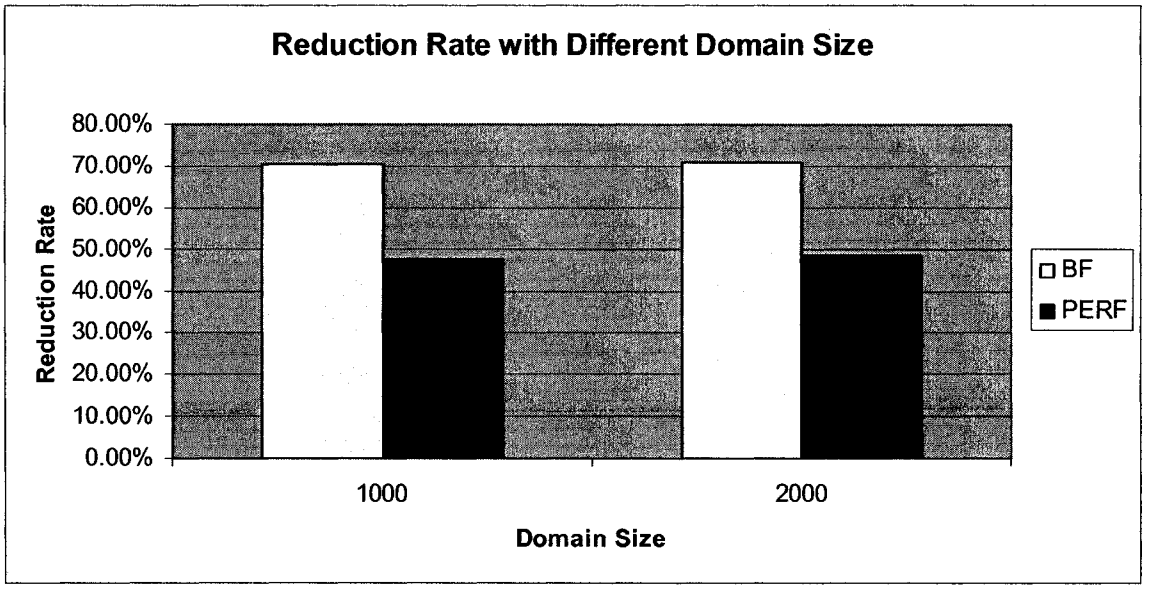

Figure 19: Reduction Rate with Different Domain Size

As it is illustrated in the experiment, from the Figure 19.We notice that, both of our algorithms performance seems to increase slightly with the growth of domain size.

4.4 Evaluation and Discussion

In our experiment platform, after Large amount of queries with arbitrary number of relations from 3 to 6, join attributes from 2 to 4, selectivity range from low to high were executed as input for our propose algorithms. The output result data show that both of Algorithm PERF and Algorithm BF make greater enhancement over the Initial Feasible Solution (IFS). By comparing with between PERF join and Bloom Filter join, we have following conclusion.

- The reduction rate of proposed algorithms increase by the selectivity level increase. In another words, with same numbers of relations and attributes, when the selectivity in the low level, our proposed algorithms have the best performance.
- The number of relations is an important factor for both of proposed algorithms. More relations the distributed query has, better reduction power it would produce.
- The number of join attributes is another important factor to affect the query performance. The query has more attributes participated will have better performance.
- The domain size doesn't affect our proposed algorithms too much. With the increase of domain size, the distributed under the same condition of selectivity, relations and join attribute, the reduction rate for both algorithms have slightly improvement.

Both our Algorithms perform better in the high level selectivity condition, while getting worse with the selectivity level getting lower. In general speaking, Algorithm BF performs better in almost all circumstance. Especially with a lower selectivity, when algorithm PERF declines its performance to a great extent. Algorithm BF can still produce a rather effective reduction rate.

A traditionally more acceptable understanding was that Perf join could perform better than bloom filter, because it has similar data storage and never encounter the information loss due to the hash function. However, in our experimental

environment, we assume that a perfect hash function has been applied which means no false drop would happen. Plus, we applied bloom filter twice on both forward reduction phase and backward reduction phase in two-way semijoin. Especially in forward reduction phase. The bloom filter based algorithm can effectively reduce the transmission cost while perf based 2-way join algorithm which still send original semijoin projection. Thus it is rather safe to conclude that two-way bloom filter based algorithm performs better than perf join.

Chapter 5 Conclusion and Future work

5.1 Conclusion

The main purpose of distributed query processing is to find the best sequence of database operation to minimize the transmission cost. Because most of implemented algorithms in this area are heuristic, our objective is to figure out the near-optimal solutions.

The main approaches in distributed query processing can be classify to 3 categories, which are join based, semijoin based and filter based. Compare with the join based algorithms always involve the large data transmission, semijoin acts as a powerful reducer in distributed query processing which only send the join attribute projection to instead of delivering the whole relation from one relation to another. As a result, the tuples which are not contributive for the query will be eliminated before they sent to final query site. Semijoin based algorithm still spend a lot even applied it's extend version 2-way semijoin by giving extra backward reduction. Filter based algorithms proposed as cheap prototype which use a bit vector to represent the semijoin projection information during the data transmission. Since bloom filter implement with hash function, the false drop caused by hash collusion can not be avoid. Perf was brought into this field as a novel solution. It use a small size bit vector (perf) based on tuples scan order to encode the join information during backward reduction phase of 2-way join. Compare with bloom filter join, it has same data storage with bloom filter and would never involve the information loss caused by hash collusion.

From above point of view, we know that the cheapest solution is to represent the original semijoin projection with a smaller size bit vector during the data transmission. In this thesis, we propose two filter-based algorithms by applying PERF and Bloom Filter technique on 2-way semijoin algorithm respectively. In algorithm PERF, we construct the perf base on tuple scan order in backward

reduction phase while with algorithm, we apply the bloom filter for semijoin projection in forward reduction phase while generate bloom filter for join match set in backward way.

After run large amount experiments with distributed queries as input. Both of algorithm PERF and algorithm BF make show their reduction power by comparing with Initial Feasible Solution. The evaluation between 2 proposed algorithms show that algorithm BF outperform algorithm PERF in most condition especially with a lower selectivity. In this kind of situation, while algorithm PERF declines its performance to a great extent. Algorithm BF can still produce a rather effective reduction rate.

Perf should outperform BF if we just compare them simply. However, in our experimental environment, we adopt perfect hash function to address tuple. Plus, we applied bloom filter twice on both forward reduction phase and backward reduction phase base on two-way semijoin. Due to the intent of perf is to reduce the transmission cost during the backward reduction. There is still expensive cost in forward reduction phase. With algorithm BF we proposed, it eliminates large transmission cost in forward phase. Even think about the false drop in tolerable range, algorithm BF can still show its good performance.

5.2 Future Work

In our experiment platform, we performed 10 queries for each kind of query over 36 query types. However, in order to get more accurate and persuasive result from experiments, we need to endeavor larger amount of query for the proposed algorithm. We also need to increase the size for the relations and domain to make the result appropriate for real-life circumstances and applications.

In Algorithm BF, to simplify the question, we assume that we use the perfect hash function which means no false drop would happen. While in the real case, collision can not be avoided in the hash filter based algorithm. So there are still some spaces to continue our research under the situation with collision and try to find optimal hash function to apply on the semijoin projection to minimize the transmission cost.

In Algorithm PERF with very high or very low semijoin selectivity, it will contain large amount of 0 or 1. We can try to compress the encode join information by sending the address of 1 or 0 but within Perf [LR95] to gain extra reduction during the transmission.

Appendix: Testing Environment

Hardware:

Dell Latitude D620

Base: Intel® Core(TM) 2 CPU T5600 @1.83 GHz

Memory: 1GB Dual Channel DDR2 SDRAM at 400MHz (4x256M)

Software:

Windows XP Professional SP2

Microsoft Visual Studio 2003

Microsoft .Net Framework 1.1

Bibliography

[AHY83] Peter M. G. Apers, Alan R. Hevner, and S. Bing Yao. "Optimization Algorithms for Distributed Queries". IEEE Transactions on Software Engineering 9(1), Pages: 57-68, 1983.

[B70] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," Comm. ACM, vol. 13(7), pp. 422-426, 1970.

[B95] Todd Bealor, "Semi-join Strategies For Total Cost Minimization In Distributed Query Processing", Master Thesis, University of Windsor, Canada, 1995.

[BC81] P.A. Bernstein and D.W. Chiu "Query Processing in a System for Distributed Databases (SDD-1)," ACM Trans. Database Syst. Vol 6, no.4, Dec 1981.

[BC81] BERNSTEIN, P., AND CHIU, D. 1981. Using semijoins to solve relational queries. J. ACM 28, I (Jan.),25-40

[BL82] P.A. Black and W.S. Luk. A new heuristic for generating semi-join programs for distributed query processing. IEEE COMPSAC, 581-588, 1982.

[BPR90] P. Boderick, J. Pyra, and J. S. Riordan. Correcting execution of distributed queries. In Proc. of 2nd Int. Symp. on Databases in Parallel and Distributed Systems. 1990.

[BR88] P. Bodorik and J.S. Riordon. A Threshold Mechanism for Distributed Query Processing. InProc. ofthe 16-th Annual ACM Computer Science Conference, pages 616{625, 1988

[BR98] P. Bodorik, J. S. Riordon "Heuristic Algorithms for Distributed Query Processing" Proc. Of the Int. Conf. on Database in Parallel and Distributed Systems, Austin, Texas, Dec 1988.

[BRJ87] P. Bodorik, J. S. Riordon and C. Jacob"Dynamic Distributed Query Processing Techniques" Technical Report, School of Computer Science, Technical Uni. Of Nova Scotia, Halifax, NS, Canada, Sept 1987

[BRP92] P. Bodorik, J. Riordo, and J. Pyra. " Deciding to Correct Distributed Query Processing " IEEE Transactions on Knowledege and Data Engineering, 4(3), June 1992

[C98] Surajit Chaudhuri "An overview of query optimization in relational systems." In Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 98, pages 34–43, 1998

[CBH84] D. Chiu, P. Bernstein, and Y. Ho, "Optimizing chain queries in a distributed database system," Siam Journal of computing, vol. 13(1), pp. 116-134, 1984.

[CDT+00] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for Internet databases. SIGMOD Record (ACM Special Interest Group on Management of Data), 29(2):379–391, 2000.

[CH84] D. Chiu and Y. Ho, "Optimizing star queries in a distributed database system," in VLDB, pp. 959-967, 1984.

[CL80] A. Chen and V. Li, "A method for interpreting tree queries into optimal semijoin expressions," in ACM SZGMOD, 1980.

[CL84] A.L.P Chen and VOK Li " Improvement Algorithms for Semijoin Query Processing Program in Distributed Database Systems " IEEE Trans. Computers C-33,11, Nov 1984

[CL89] J. K. AHN and S. C. MOON " Optimization Joins between Two Fragmented Relations on a Broadcast Local Network" IEEE-TSE 15(1), 26-38 (1989)

[CL90] L. Chen and V. Li, "Domain-Specific Semijoin: A New Operation for Distributed Query Processing" Info. Sci., vol. 52, pp. 165-183, 1990.

[CP84] S. Ceri and G. Pelagatti. Distributed Database Design: Principles and Systems. MacGraw-Hill (New York NY), 1984

[CP84] S. Ceri and G. Pelagatti. Distributed Databases: Principles and Systems. McGraw-Hill, 1984.

[CS94] S. Chaudhuri and K. Shim Including Group-By in Query Optimization. In Proc. of VLDB, Santiago. 1994.

[CY90] M. S. Chen and P. S. Yu "Using Join Operations as Reducers in Distributed Query Processing" - DPDS, 1990 - ieeexplore.ieee.org

[CY90] M.S. Chen and P.S. Yu, " Using Combination of Join and Semijoin Operations for Distributed Query Processing " Processing of the 10th intern'l Conf. on Distributed Computing Systems, pp. 328-335, May 1990

[CY91] M.-S. Chen and P. S. Yu. "Determining Beneficial Semijoins for a Join Sequence in Distributed Query Processing" Proceedings of the 7th Intern'1 Conf. on Data Engineering, pages 50-58, April 1991. To appear in IEEE Trans. on Parallel and Distributed Systems.

[DG92] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. Communications of the ACM, 35(6):85–98, June 1992.

[G93] G. Graefe. Query Evaluation Techniques for Large Databases. In ACM Computing Surveys: Vol25. No 2.. June 1993.

[GD81] M. G. Gouda and U. Dayal "Optimal Semijoin Schedules For Query Processing in Local Distributed Database Systems" - SIGMOD Conference, 1981 - portal.acm.org

[GGS96] S. GANGULY, A. GOEL AND A. SILBERSCHATZ, Efficient and accurate cost models for parallel query optimization. In Proceedings of the ACM SIGMOD/SIGACT Conference on Principles of Database Systems (PODS) (Montreal, Canada, June), 172–181. 1996.

[GW89] G. Graefe and K. Ward, "Dynamic query evaluation plans," in ACM SIGMOD, pp- 358-366, 1989.

[H80] A. Hevner, The optimization of query processing in distributed database systems. PhD thesis, Perdue University, 1980.

[HCY94] H. I. Hsiao, M. S. Chen, P. S. Yu "On Parallel Execution of Multiple Pipelined Hash Joins", SIGMOD Conf., Minneapolis, 1994

[HF01] R. A. Haraty and R. C. Fany" Query Acceleration in Distributed Database Systems " 2001

[HFB00] RA Haraty, RC Fany, L Beirut " Distributed Query Optimization Using PERF Joins " SAC (1), 2000 - portal.acm.org

[HKW+97] L. HAAS, D. KOSSMANN, E. WIMMERS AND J. YANG, Optimizing queries across diverse data sources. In Proceedings of the Conference on Very Large Data Bases (VLDB) (Athens, Greece, Aug.), 276–285. 1997.

[HWY85] A. R. Hevner, O. Q. Wu, and S. B. Yao. Query optimization on local area networks. ACM Transactions January 1985. CODEN ATOSDO.

[KR87] H. Kang and N. Roussopoulos " Using 2-way Semijoins in Distributed Query Processing " in Proceedings of the Third International Conference on Data Engineering, 1987. PP. 644-651

[KYY83] Y. Kambayaashi, M. Yoshikawa, and S. Yajima. Query Processing for Distributed Databases Using Generalized Semi-join. In ACM SIGMOD International Conference on Management of Data, San Jose, CA, May 1983.

[LMH+85] G. Lehman, C. Mohan, L. Haas, D. Daniels, B. Lindsay, P. Selinger and P. Wilms. Query Processing in R*. In Query Processing in Database Systems Springer Verlag, 1985.

[LPP91] P. Legato, G. Paletta, and L. Palopoli " Optimization of Join Strategies in Distributed Databases" Information Systems, 16(4):363{374, 1991

[LR95] Z. Li and K. Ross "PERF Join: An Alternative to Two-way Semijoin and Bloomjoin" Technical Report. Columbia University, New York. 1995

[M01] M. Mitzenmacher "Compressed Bloom Filters" In Twenti-eth ACM Symposium on Principles of Distributed Computing (PODC 2001), 2001.

[M83] J. Mullin, "(1983) A second look at bloom filters," Comm. ACM, vol. 26(8), pp. 570-541, 1983.

[M90] J.K. Mullin "Optimal semijoins for distributed database systems", IEEE Transactions on Software Engineering 16 (1990).

[M93] James K. Mullin "Estimating the Size of a Relational Join" Information Systems, 18(3):189 – 196, 1993. ISSN 0306-4379

[M96] J. M. Morrissey "Reduction Filters for Minimizing Data Transfers in Distributed Query Optimization" In Proceedings of CCECE'96, Calgary, May,1996

[MB95] J. M. Morrissey and S. Bandyopadhyay "Computer communication technology and its effects on distributed query optimization strategies" Electrical and Computer Engineering, 1995. Canadian Conference on Volume: 1 5-8 Sep 1995

[MB97] J. Morrissey and W. Bealor, "Minimizing data transfers in distributed query processing: a comparative study and evaluation," The Computer Journal, vol. 39(8), 1997.

[ML96] L.MACKERT AND G. LOHMAN, R* optimizer validation and performance evaluation for distributed queries. In Proceedings of the Conference on Very Large Data Bases (VLDB) (Kyoto, Japan), 149–159. 1986.

[MO97] J. Morrissey and W. Osborn, "Experiments with the use of reduction filters in distributed query optimization," in Proceedings of the 9th IASXED International Conference on Parallel and Distributed Systems, (Georgetown University, Washington),pp. 327-330, 1997.

[MO98] J. M. Morrissey and W. K. Osborn "Distributed query optimization using reduction filters" Electrical and Computer Engineering, 1998. IEEE Canadian Conference on Volume: 2 24-28 May 1998

[MO99] J. M. Morrissey and O. Ogunbadejo "Combining semijoins and hash-semijoins in a distributed query processing strategy" Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering Shaw Conference Center, Edmonton, Alberta, Canada May 9-12 1999

[MO99] J. M. Morrissey and W. K. Osborn "The effect of collisions on the performance of reduction filters" Electrical and Computer Engineering, 1999 IEEE Canadian Conference on Volume: 1 1999

[MOL00] J.M. Morrissey, W.K. Osborn, Y. Liang "Collisions and reduction filters in distributed query processing" CAN CONF ELECTR COMPUT ENG, 2000 - ieeexplore.ieee.org

[O92] M. Orlowski " On Optimisation of Joins in Distributed Database System " Technical Report1/92, School of Information Systems, Faculty of Information Technology, Queensland University of Technology, January 1992

[O98] W. Osborn, "The use of reduction filters in distributed query optimization," Master's thesis, The University of Windsor, 1998.

[OV91] M. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems. Prentice-Hall, 1991.

[PC90] W. Perrizo and C. Chen "Composite Semijoins in Distributed Query Processing" Information Science, March. 1990

[PIH96+] V. Poosala, Y. Ioannidis, P. Harts and E. Shekita. Improved Histograms for Selectivity Estimation. In Proc. of ACM SIGMOD, Montreal, Canada 1996.

[Q88] G. Z. Qadah, Filter-based join algorithm on uniprocessor and distributed memory multiprocessor database machines, vol. 303 of Lecture Notes in Computer Science, pp. 388-413. Springer-Verlag, 1988.

[RK91] N. Roussopoulos and H. Kang " A Pipeline N-way Join Algorithm Based on the 2-way Semijoin Program " IEEE Trans, on Knowledge and Data Engineering, Vol. 3, No. 4, Dec. 1991, pp. 486-495

[TC92] J. Tseng and A. P. Chen " Improving Distributed Query Processing by Hash-Semijoins " Journal of Information Science and Engineering, 1992

[URJ+83] D. Jantz, E. A. Unger, R. McBride, and J. Slonim. Query processing in a distributed data base. In Proceedings of the 1983 ACM SIGSMALL symposium on Personal andsmall computers, pages 237–244, 1983

[V02] Patrick Valduriez "Principles of Distributed Database Systems" (Second Edition). ISBN 7-302-05493-2/TP.3230. 2002

[VG84] P. Valduriez and G. Gardarin, "Join and semijoin algorithms for a multiprocessor database machine," ACM Transactions on database system, pp. 133-161, 1984.

[WC93] C. Wang and M. Chen, "On the complexity of distributed query optimization," tech. rep., IBM Technical Report RC 18671, 1993.

[WLC91] C. Wang, V. Li, and A. Chen, " Distributed Query Optimization by One-Shot Fixed-Precision Semijoin Execution" in Proc. 7th Znt. Con$ on Data Engineering, pp. 756-763, 1991

[Yang05] Li.Yang. "An Evaluation of PERF Joins for a Two-Way Semijoin Based Algorithm". Master thesis, University of Windsor, 2005

[YC83] C. T. YU AND C. C. CHANG "On the design of a query processing strategy in a distributed database environment". In SIGMOD 83, Proceedings of the Annual Meeting (San Jose, Calif., May 23-26). ACM, New York, pp. 30-39. 1983.

[YC84] C. T. Yu and C. C. Chang. Distributed query processing. ACM Computing Surveys, 16(4):399–433, Dec.1984.

[YM98] C. Yu and W. Meng. Principles of Database Query Processing for Advanced Applications. Morgan Kauf-mann, San Francisco, 1998.

Vita Auctoris

Name: Ming Pei

PLACE OF BIRTH: Wuhan, Hubei, China

YEAR OF BIRTH: 1976

EDUCTION: M.Sc., Computer Science,
University of Windsor,
Windsor, Ontario, Canada
2004 ~ 2007

B.Sc., Computer Science,
Navy University of Engineering,
Wuhan, Hubei, China
1994 ~ 1998